Final Report for Period: 09/2007 - 08/2008
Submitted on: 12/20/2008
Principal Investigator: Harrold, Mary J.
Award ID: 0429117
Organization: GA Tech Res Corp - GIT
Submitted By:
Harrold, Mary - Principal Investigator
Title:
HDCCSR: Software Self-Awareness Using Dynamic Analysis and Markov Models

Project Participants

Senior Personnel

Name: Harrold, Mary
Worked for more than 160 Hours: Yes
Contribution to Project:

Name: Rugaber, Jon Spencer
Worked for more than 160 Hours: Yes
Contribution to Project:

Name: Rehg, James
Worked for more than 160 Hours: Yes
Contribution to Project:

Post-doc

Graduate Student

Name: Bowring, James
Worked for more than 160 Hours: Yes
Contribution to Project:
Jim is was the initial main student researcher on this project and has created the algorithms and infrastructure for the behavior classification. The work described in the Activities and Findings describes this work.

He has applied his research work to fault localization, and that work is described in the Activities and Findings document.

Name: Sharma, Gaurav
Worked for more than 160 Hours: Yes
Contribution to Project:
Gaurav created the infrastructure that was required for the work on value-flow analysis---which we call data bins. He created this by modifying a tool called Daikon, created and maintained by Mike Ernst and his group at MIT. His work facilitated the experiments on data-bin classifiers.

Name: Baah, George
Worked for more than 160 Hours: Yes
Contribution to Project:
George has been researching ways to find anomalous behavior and its
cause for continuously-running programs. He has developed an on-line algorithm that will detect anomalous behavior during execution, based on a model of the system, and then help to identify the cause of the anomalous behavior (e.g., the fault).

He has also developed algorithms to provide statistical models that can be used for fault diagnosis

Name: Kishinovski, Irina

Worked for more than 160 Hours: Yes

Contribution to Project:

Irina has worked on two aspects of the project. The first uses the behavior classification to create test suites for legacy software. The second collects data for an experiment that will determine how close the behaviors recognized by our classifiers relate to the behaviors identified in the requirements specification.

Name: Jones, James

Worked for more than 160 Hours: Yes

Contribution to Project:

Jim has worked on the clustering for fault localization for the project.

Name: Navarro, Angela

Worked for more than 160 Hours: Yes

Contribution to Project:

Masters student researcher; worked on ISVis Adaptation project

Name: Calvert, Peter

Worked for more than 160 Hours: Yes

Contribution to Project:

Masters student researcher; worked on ISVis Adaptation project

Name: Biddy, Michele

Worked for more than 160 Hours: Yes

Contribution to Project:

Masters student researcher; worked on ISVis Adaptation project

Name: Norton, Phil

Worked for more than 160 Hours: Yes

Contribution to Project:

Masters student researcher; worked on ISVis Adaptation project

Name: Huang, Chung-Yen

Worked for more than 160 Hours: Yes

Contribution to Project:

Masters student researcher; worked on ISVis Adaptation project

Name: Bernard, Andrew

Worked for more than 160 Hours: No

Contribution to Project:

Masters student researcher; worked on ISVis Adaptation project

Undergraduate Student

Name: Jensen, Allison

Worked for more than 160 Hours: Yes

Contribution to Project:

Allison has been investigating the location of patterns (motifs) that will characterize behaviors.

Name: Hazell, Greig

Worked for more than 160 Hours: Yes

Contribution to Project:

Greig has been investigating the location of patterns (motifs) that will characterize behaviors.

Technician, Programmer

Other Participant

Research Experience for Undergraduates

Organizational Partners

NASA Ames Intelligent Systems Division
Will make testbeds available.

Other Collaborators or Contacts

Tata Consultancy Services
Reflective Corporation

Activities and Findings

Research and Education Activities: (See PDF version submitted by PI at the end of the report)
See attached 'Activities and Findings' document.

Findings: (See PDF version submitted by PI at the end of the report)
See attached 'Activities and Findings' document.

Training and Development:
Those who have worked on the project have gained research skills in program analysis, machine learning, and software engineering. They have also gained experience with using both C and Java, as well as environments such as dot Net.

Jim Bowring completed his Ph.D. and is an assistant professor at the College of Charleston.

Jim Jones completed his Ph.D. and is an assistant professor at University of California Irvine.

George Baah completed his qualifier exam.

Outreach Activities:
Participated in conferences and workshops.

Journal Publications

Books or Other One-time Publications

James F. Bowring, James M. Rehg, and Mary Jean Harrold, "Active learning for automatic classification of software behavior", (2004).
Proceedings, Published
Editor(s): ACM Press
Collection: ISSTA '04: Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis 2004
Bibliography: pages 195-205

James F. Bowring, Mary Jean Harrold, James M. Rehg, "Improving the classification of software behaviors using ensembles of control-flow and data-flow classifiers", (2005). Technical report, Published
Editor(s): Technical report, under review
Bibliography: 10 pages

George Baah, Alexander Gray, and Mary Jean Harrold, "On-line Anomaly Detection of Deployed Software: A Statistical Machine Learning Approach", (2006). Conference Proceedings, Published
Collection: Proceedings
Bibliography: Proceedings of the Third International Workshop on Software Quality Assurance (SOQUA 2006)

James F. Bowring, "Modeling and Predicting Software Behaviors", (2006). Thesis, Published
Collection: Dissertation
Bibliography: Ph.D. Dissertation, December 2006

James A. Jones, James F. Bowring, and Mary Jean Harrold, "Debugging in Parallel", (2007). Conference Proceedings, Published
Collection: Conference Proceedings
Bibliography: 2007 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2007), pages 16-26, July 2007.

George K. Baah, Andy Podgurski, and Mary Jean Harrold, "The Probabilistic Program Dependence Graph and Its Application to Fault Diagnosis", (2008). Conference Proceedings, Published
Collection: Conference Proceedings
Bibliography: Proceedings of the 2008 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), pages 189--200, July 2008.

Web/Internet Site

Other Specific Products

Product Type:

Software (or netware)

Product Description:

We have developed an infrastructure on which many experiments can be performed for behavior classification. To date, we have used this software for building behavior classifiers that have been used for experiments in test-suite augmentation, remote monitoring, and fault localization.

Sharing Information:

We have made this software available withing Georgia Tech to other researchers.

Contributions

Contributions within Discipline:

Developed new framework for developing, implementing, and evaluating new program behavior classifiers.
Developed system called Gimbal to implement this framework.
Extended work on behavior modeling and classification.
Created new techniques using machine learning and program analysis that create models and classifiers.
Applied behavior classification and clustering to the problem of fault-localization.
Developed techniques for failure identification and fault-localization for long-running (or continuous) programs that predict failures as the program is executing, and help in localizing the fault or anomoly that is occurring.

Contributions to Other Disciplines:

This project is developing new or adapting existing machine learning techniques and applying them to software-engineering tasks, particularly tasks of failure identification and fault-localization.

Contributions to Human Resource Development:

Support for one M.S. student who completed his project and graduated in May 2005, and is now working at Medtronics in Minneapolis.

Support for one Ph.D. student who worked on the project, completed his Ph.D. in August 2006, and is an assistant professor at the College of Charleston (SC)..

Support for a second Ph.D. student (a minority student) who is working on the project. He completed his qualifier exam in 2006, and will present his Ph.D. proposal in 2009.

Support for a third Ph.D. student who completed his dissertation, graduated in 2008, and is an assistant professor at University of California, Irvine.

Supported two undergraduate students---both from underrepresented groups---for summer 2005 who learned research and development skills. Both of these students continued working on the project in AY2005-06. One of the students graduated and is working in Atlanta. The other student is working on the project for summer 2006, and decided, because of his experience with research, to pursue a graduate degree after he graduated in May 2007.

Training for these students in program analysis, machine learning, software engineering, experimental methods, development environments, oral and communication skills.

Contributions to Resources for Research and Education:

Contributions Beyond Science and Engineering:

We have submitted preliminary invention disclosures to Georgia Tech for the research performed within the project.

Categories for which nothing is reported:

Any Journal

Any Web/Internet Site

Contributions: To Any Resources for Research and Education

Research and Education Activities and Findings
December 2008
Final Report

Activities

For discussion of our activities, this section contains two subsections. The first subsection describes activities related to behavior classification when the program has completed—either terminated or crashed. These techniques use information about the complete execution for their computations. The second subsection describes activities related to behavior classification (and then fault localization) while the program is running. These techniques can flag anomalis as the program is executing.

Behavior classification for complete program executions

We have implemented a research framework to investigate the predictive power of specific features of program executions and to lay the groundwork for the Tripwire technology. The framework consists of the Aristotle Analysis [1] system for C programs plus a new component, Argo. With the framework we can first select a subject program, instrument it to collect profiles of specific state-transition features, execute it with specific inputs mapped to known behaviors such as pass and fail, build stochastic models of the data. Second, the framework supports machine-learning algorithms such as active learning with which to we can train behavior classifiers based on these models. Finally, with the framework, we can empirically evaluate the classifiers by measuring their classification rates on additional executions of the subject program. Third, with our framework, we can evaluate ensemble classifiers build from classifiers of two or more individual features.

We have investigated three individual control-flow features: branch profiles, branch-to-branch profiles, and method caller/callee profiles [3]. We have chosen for efficiency to model the data as Markov chains, by taking the Markov assumption that only the current state of a model predicts the next state. We invented a novel summary for data-flows to efficiently capture aspects of data behaviors. For each variable of interest, a databin partitions the range of values into a fixed and small number of bins (this approach is similar to that used in histograms) [4]. These bins are percentiles of the range for a given variable, and each percentile becomes a state in a transition matrix. This stochastic construct is also a Markov chain and thus databins are a data-flow feature we investigated with our framework.

We have investigated the use of these behavior models for several applications. The first application is the problem of building up effective test suites for programs in an efficient way from an initial set of test cases for the program—we call this the *test-suite augmentation problem*. Our technique for test-suite augmentation uses an initial test suite for the program for which we had behavior labels (e.g., "pass" and "fail") to construct a classifier using the approach described above. Our technique then uses this classifier along with a random test-case generator for the program. For each new test case randomly generated, the system uses the classifier to predict whether the

behavior, modeled by the data gathered during the programs' execution, is one that the classifier recognizes. If the classifier recognizes the behavior (i.e., this behavior is already represented in the test suite), it discards the test case as one not needed for the testing. If the classifier does not recognize the behavior, then a developer needs to assess the outcome and label the execution. This new test case is added to the test suite and used to rebuild the classifier, and the process continues. The advantage of this approach is that hand-labeling of many test cases is avoided because the classifier filters out those test cases that are already represented in the test suite. We have published initial results of using the classifiers for test-suite augmentation [3], have done additional experiments on larger subject programs. The results are reported in Jim Bowring's dissertation [5], and we are currently writing a journal paper that will report the results.

A second application of this approach is for building test suites for legacy software. This problem was presented to us by one of our industry partners—TCS. In this case, the software has been running (mostly correctly) for some number of years, but now needs to be ported to new machines, languages, etc. Our approach is to monitor the live execution of the system, gather test inputs, and use these as test cases to build our initial models of the system. We then use a technique similar to test-suite augmentation except that instead of randomly generating test cases for inputs, we continue to monitor the live system, and when new behaviors are found, we add them to the legacy test suite. We are also investigating what kind of coverage we should recommend to get a good test suite using this approach.

A third application is the problem of clustering test cases that represent like behaviors related to faults for use in fault-localization. In previous work, we have presented a fault-localization technique that uses statement-coverage information about a test suite along with pass/fail results of the execution of test cases in the test suite, to identify and order statements according to their suspiciousness. Of course, the developer does not know how many faults (bugs, errors) are in the program, but to reduce the time to fix faulty software, may want to identify subsets of test cases in the test suite that seem to be associated with different faults. This would let additional developers work on parts of the software related to these subsets of test cases to (1) fix the bugs faster by performing the fault-localization in parallel and (2) to reduce "noise" that may be in the results when all test cases are used for the analysis. To date, we have performed experiments with this approach, and will present the results and the reduction that can be achieved using the approach [6].

The fourth application is the problem of fault detection. We have used runtime information about data and control dependencies and program state to create a new program representation, which we call the *Probabilistic Program Dependence Graph* or PPDG[7]. We have used the PPDG for fault localization and fault diagnosis.

For all these techniques described above, we are implementing tools, and gathering and preparing subjects.

Behavior classification for executing programs

During this past year, we continued our other component of this research—how to recognize behavior while the program is executing, and flag anomalous or faulty behavior. For this project, we have been considering other types of information about the executing program such as states of the predicates during execution. Using this information, like our previous approach, uses machine learning to build models of the execution. However, we have investigated the use of other machine-learning algorithms and are currently using Hidden Markov Models. Like the previous approach, we use a set of test cases to build the models for classification. However, unlike previous approaches, we monitor the execution, and since this approach doesn't need the entire execution, can consider the behavior seen so far and predict behavior. If the behavior is found to be anomalous in some part

of the program, the technique can also suggest the locations in which to search for the anomaly.

The advantage of this approach is that it could be applied to long-running programs or programs that run continuously, and faults could be found early, possibly before they cause serious problems.

ISVis Adaption Project

The goal of the project is to study the effort involved in revitalizing legacy software. ISVis is a reverse engineering tool that combines static and dynamic analysis in support of architecture localization. It was developed as part of a DARPA project that took place in the last half of the 1990s. The program consisted of 23KLOC of C++ code, and was not runable due to the following environmental changes.

1. The version of C++ in which it was written had changed significantly from the pre-ANSI standard version in which it was originally written.
2. The version of the Solaris operating system for which it was originally written had changed.
3. The original version depended on a third-party library, Rogue Wave, to provide persistence. This package was no longer available to us.
4. The original version used the X-Windows Motif widget library, which had undergone changes.
5. The original version targeted specific display technology that had been superceded. Moreover, the display-dependent code was heavily interleaved with the functional code.

The project has proceeded as a series of six, nine-credit Master's projects. Each participant has tracked his/her effort and code changes, so we have comprehensive empirical data on the effort. So far, the first four items above have been successfully dealt with. The code is now ANSI-C++ compliant and compatible with the current versions of Solaris and the Motif library. The Rogue Wave code has been replaced by a combination of the Standard Template Library and the Boost persistence package.

The sixth Master's project, addressing item five above, is currently underway. Its goal is to isolate and re-architect the display-dependent code so that a platform-independent package, such as FLTK, can be substituted.

Findings

In this section, we also divide the findings into two subsections. Each subsection reports what we have found so far on that part of the research.

Behavior classification for complete program executions

We evaluated the four program features (branch profiles, branch-branch profiles, metnod-method profiles, and databin profiles) individually and in some cases, jointly, for ten small and medium sized C programs in our subject library. We found that of the control-flow features, the branch feature generally outperformed the branch-to-branch feature and the method caller/callee feature [3]. The reason is that the branch feature subsumes the method caller/callee feature and is the core of the branch-to-branch feature. However, there were two instances where the control-flow features performed poorly. The reason is that the subjects lacked differentiating control patterns. The databin feature worked well for six subjects, but never out-performed the branch feature [4]. We

evaluated ensembles of classifiers built from the branch feature and the databin feature. These empirical studies show that ensemble classifiers can yield a classification rate superior to or as good as either component classifier. This finding of improved classification rates for ensembles suggests that control-flow and data-flow features of program executions can capture divergent statistical views of behaviors for some subjects. These studies also validate the usefulness of databin modeling and further demonstrate that, as with control-flow features, active learning is more efficient and effective than the conventional batch learning for databin profiles. These studies also that show that, for our subject programs, the branch-profile feature has up to 10% more predictive power than the databin profiles. However, the competitive performance of the databin classifiers suggests that data-flow features may be as important as control-flow features in characterizing program behavior. We are gaining an understanding of individual features so that we might better understand how to combine them to best effect.

For the three applications of our behavior classifier, we have some results. For the first application—test-suite augmentation—we performed initial experiments on the set of subjects described above, and added test cases to the test suite. Subsequent to the publication in ISSTA 2004, we have performed additional experiments on larger subjects. These results are reported in Jim Bowring's dissertation [5], and we are preparing a journal paper that describes the technique and this application in more detail and includes these experiments.

For the second application, performed experiments and showed that the technique builds good legacy test suites. We are also investigating ways to perform this experiment on real systems in use.

For the third application, we have completed a set of experiments that demonstrate that we can use the behavior classifier to cluster test cases. We have found that the clusters help in quicker location of the faults. We are also investigating how to assign these clusters to groups of developers, based on the apparent components of the system that appear to be involved in the fault. We reported the results of these findings at ISSTA 2007 [6].

Behavior classification for executing programs

We reported the results of this work in a paper at SOQUA in November 2006 [2]. To date, we have shown that it can identify anomalies and help in locating faults. That work let us understand better the problem, and we are now investigating the use of better models—ones based on Baysian netrorks.

We extended this approach for the creation of the PPDG (described above) and have performed experiments. Our experiments show that the PPDG can provide better fault localization than existing techniques [7].

ISVis Adaption Project

As the project is still underway, and no papers have yet been written.

References

[1] Aristotle Research Group. ARISTOTLE: Software engineering tools, 2002. `http://www.cc.gatech.edu/aristotle/`.

[2] George Kofi Baah, Alexander Gray, and Mary Jean Harrold. On-line Anomaly Detection of Deployed Software: A Statistical Machine Learning Approach. Proceedings of the Third In-

ternational Workshop on Software Quality Assurance (SOQUA 2006), pages 70–77, November 2006.

[3] James F. Bowring, James M. Rehg, and Mary Jean Harrold. Active learning for automatic classification of software behavior. Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004), pages 195–205, July 2004.

[4] James F. Bowring, James M. Rehg, and Mary Jean Harrold. Improving the classification of software behaviors using ensembles of control-flow and data-flow classifiers. Technical Report Technical Report GIT-CERCS-05-10, College of Computing, Georgia Institute of Technology, 2005.

[5] James F. Bowring. Modeling and Predicting Software Behaviors. Ph.D. Dissertation (advisors: Mary Jean Harrold and James M. Rehg), December 2006.

[6] James A. Jones, James F. Bowring, and Mary Jean Harrold. Debugging in Parallel. Proceedings of the 2007 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2007), pages 16-26, July 2007.

[7] George K. Baah, Andy Podgurski, and Mary Jean Harrold. The Probabilistic Program Dependence Graph and Its Application to Fault Diagnosis Proceedings of the 2008 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), pages 189–200, July 2008.

Research and Education Activities and Findings
December 2008
Final Report

Activities

For discussion of our activities, this section contains two subsections. The first subsection describes activities related to behavior classification when the program has completed—either terminated or crashed. These techniques use information about the complete execution for their computations. The second subsection describes activities related to behavior classification (and then fault localization) while the program is running. These techniques can flag anomalis as the program is executing.

Behavior classification for complete program executions

We have implemented a research framework to investigate the predictive power of specific features of program executions and to lay the groundwork for the Tripwire technology. The framework consists of the Aristotle Analysis [1] system for C programs plus a new component, Argo. With the framework we can first select a subject program, instrument it to collect profiles of specific state-transition features, execute it with specific inputs mapped to known behaviors such as pass and fail, build stochastic models of the data. Second, the framework supports machine-learning algorithms such as active learning with which to we can train behavior classifiers based on these models. Finally, with the framework, we can empirically evaluate the classifiers by measuring their classification rates on additional executions of the subject program. Third, with our framework, we can evaluate ensemble classifiers build from classifiers of two or more individual features.

We have investigated three individual control-flow features: branch profiles, branch-to-branch profiles, and method caller/callee profiles [3]. We have chosen for efficiency to model the data as Markov chains, by taking the Markov assumption that only the current state of a model predicts the next state. We invented a novel summary for data-flows to efficiently capture aspects of data behaviors. For each variable of interest, a databin partitions the range of values into a fixed and small number of bins (this approach is similar to that used in histograms) [4]. These bins are percentiles of the range for a given variable, and each percentile becomes a state in a transition matrix. This stochastic construct is also a Markov chain and thus databins are a data-flow feature we investigated with our framework.

We have investigated the use of these behavior models for several applications. The first application is the problem of building up effective test suites for programs in an efficient way from an initial set of test cases for the program—we call this the *test-suite augmentation problem*. Our technique for test-suite augmentation uses an initial test suite for the program for which we had behavior labels (e.g., "pass" and "fail") to construct a classifier using the approach described above. Our technique then uses this classifier along with a random test-case generator for the program. For each new test case randomly generated, the system uses the classifier to predict whether the

behavior, modeled by the data gathered during the programs' execution, is one that the classifier recognizes. If the classifier recognizes the behavior (i.e., this behavior is already represented in the test suite), it discards the test case as one not needed for the testing. If the classifier does not recognize the behavior, then a developer needs to assess the outcome and label the execution. This new test case is added to the test suite and used to rebuild the classifier, and the process continues. The advantage of this approach is that hand-labeling of many test cases is avoided because the classifier filters out those test cases that are already represented in the test suite. We have published initial results of using the classifiers for test-suite augmentation [3], have done additional experiments on larger subject programs. The results are reported in Jim Bowring's dissertation [5], and we are currently writing a journal paper that will report the results.

A second application of this approach is for building test suites for legacy software. This problem was presented to us by one of our industry partners—TCS. In this case, the software has been running (mostly correctly) for some number of years, but now needs to be ported to new machines, languages, etc. Our approach is to monitor the live execution of the system, gather test inputs, and use these as test cases to build our initial models of the system. We then use a technique similar to test-suite augmentation except that instead of randomly generating test cases for inputs, we continue to monitor the live system, and when new behaviors are found, we add them to the legacy test suite. We are also investigating what kind of coverage we should recommend to get a good test suite using this approach.

A third application is the problem of clustering test cases that represent like behaviors related to faults for use in fault-localization. In previous work, we have presented a fault-localization technique that uses statement-coverage information about a test suite along with pass/fail results of the execution of test cases in the test suite, to identify and order statements according to their suspiciousness. Of course, the developer does not know how many faults (bugs, errors) are in the program, but to reduce the time to fix faulty software, may want to identify subsets of test cases in the test suite that seem to be associated with different faults. This would let additional developers work on parts of the software related to these subsets of test cases to (1) fix the bugs faster by performing the fault-localization in parallel and (2) to reduce "noise" that may be in the results when all test cases are used for the analysis. To date, we have performed experiments with this approach, and will present the results and the reduction that can be achieved using the approach [6].

The fourth application is the problem of fault detection. We have used runtime information about data and control dependencies and program state to create a new program representation, which we call the *Probabilistic Program Dependence Graph* or PPDG[7]. We have used the PPDG for fault localization and fault diagnosis.

For all these techniques described above, we are implementing tools, and gathering and preparing subjects.

Behavior classification for executing programs

During this past year, we continued our other component of this research—how to recognize behavior while the program is executing, and flag anomalous or faulty behavior. For this project, we have been considering other types of information about the executing program such as states of the predicates during execution. Using this information, like our previous approach, uses machine learning to build models of the execution. However, we have investigated the use of other machine-learning algorithms and are currently using Hidden Markov Models. Like the previous approach, we use a set of test cases to build the models for classification. However, unlike previous approaches, we monitor the execution, and since this approach doesn't need the entire execution, can consider the behavior seen so far and predict behavior. If the behavior is found to be anomalous in some part

of the program, the technique can also suggest the locations in which to search for the anomaly.

The advantage of this approach is that it could be applied to long-running programs or programs that run continuously, and faults could be found early, possibly before they cause serious problems.

ISVis Adaption Project

The goal of the project is to study the effort involved in revitalizing legacy software. ISVis is a reverse engineering tool that combines static and dynamic analysis in support of architecture localization. It was developed as part of a DARPA project that took place in the last half of the 1990s. The program consisted of 23KLOC of C++ code, and was not runable due to the following environmental changes.

1. The version of C++ in which it was written had changed significantly from the pre-ANSI standard version in which it was originally written.
2. The version of the Solaris operating system for which it was originally written had changed.
3. The original version depended on a third-party library, Rogue Wave, to provide persistence. This package was no longer available to us.
4. The original version used the X-Windows Motif widget library, which had undergone changes.
5. The original version targeted specific display technology that had been superceded. Moreover, the display-dependent code was heavily interleaved with the functional code.

The project has proceeded as a series of six, nine-credit Master's projects. Each participant has tracked his/her effort and code changes, so we have comprehensive empirical data on the effort. So far, the first four items above have been successfully dealt with. The code is now ANSI-C++ compliant and compatible with the current versions of Solaris and the Motif library. The Rogue Wave code has been replaced by a combination of the Standard Template Library and the Boost persistence package.

The sixth Master's project, addressing item five above, is currently underway. Its goal is to isolate and re-architect the display-dependent code so that a platform-independent package, such as FLTK, can be substituted.

Findings

In this section, we also divide the findings into two subsections. Each subsection reports what we have found so far on that part of the research.

Behavior classification for complete program executions

We evaluated the four program features (branch profiles, branch-branch profiles, metnod-method profiles, and databin profiles) individually and in some cases, jointly, for ten small and medium sized C programs in our subject library. We found that of the control-flow features, the branch feature generally outperformed the branch-to-branch feature and the method caller/callee feature [3]. The reason is that the branch feature subsumes the method caller/callee feature and is the core of the branch-to-branch feature. However, there were two instances where the control-flow features performed poorly. The reason is that the subjects lacked differentiating control patterns. The databin feature worked well for six subjects, but never out-performed the branch feature [4]. We

evaluated ensembles of classifiers built from the branch feature and the databin feature. These empirical studies show that ensemble classifiers can yield a classification rate superior to or as good as either component classifier. This finding of improved classification rates for ensembles suggests that control-flow and data-flow features of program executions can capture divergent statistical views of behaviors for some subjects. These studies also validate the usefulness of databin modeling and further demonstrate that, as with control-flow features, active learning is more efficient and effective than the conventional batch learning for databin profiles. These studies also that show that, for our subject programs, the branch-profile feature has up to 10% more predictive power than the databin profiles. However, the competitive performance of the databin classifiers suggests that data-flow features may be as important as control-flow features in characterizing program behavior. We are gaining an understanding of individual features so that we might better understand how to combine them to best effect.

For the three applications of our behavior classifier, we have some results. For the first application—test-suite augmentation—we performed initial experiments on the set of subjects described above, and added test cases to the test suite. Subsequent to the publication in ISSTA 2004, we have performed additional experiments on larger subjects. These results are reported in Jim Bowring's dissertation [5], and we are preparing a journal paper that describes the technique and this application in more detail and includes these experiments.

For the second application, performed experiments and showed that the technique builds good legacy test suites. We are also investigating ways to perform this experiment on real systems in use.

For the third application, we have completed a set of experiments that demonstrate that we can use the behavior classifier to cluster test cases. We have found that the clusters help in quicker location of the faults. We are also investigating how to assign these clusters to groups of developers, based on the apparent components of the system that appear to be involved in the fault. We reported the results of these findings at ISSTA 2007 [6].

Behavior classification for executing programs

We reported the results of this work in a paper at SOQUA in November 2006 [2]. To date, we have shown that it can identify anomalies and help in locating faults. That work let us understand better the problem, and we are now investigating the use of better models—ones based on Baysian netrorks.

We extended this approach for the creation of the PPDG (described above) and have performed experiments. Our experiments show that the PPDG can provide better fault localization than existing techniques [7].

ISVis Adaption Project

As the project is still underway, and no papers have yet been written.

References

[1] Aristotle Research Group. ARISTOTLE: Software engineering tools, 2002. `http://www.cc.gatech.edu/aristotle/`.

[2] George Kofi Baah, Alexander Gray, and Mary Jean Harrold. On-line Anomaly Detection of Deployed Software: A Statistical Machine Learning Approach. Proceedings of the Third In-

ternational Workshop on Software Quality Assurance (SOQUA 2006), pages 70–77, November 2006.

[3] James F. Bowring, James M. Rehg, and Mary Jean Harrold. Active learning for automatic classification of software behavior. Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004), pages 195–205, July 2004.

[4] James F. Bowring, James M. Rehg, and Mary Jean Harrold. Improving the classification of software behaviors using ensembles of control-flow and data-flow classifiers. Technical Report Technical Report GIT-CERCS-05-10, College of Computing, Georgia Institute of Technology, 2005.

[5] James F. Bowring. Modeling and Predicting Software Behaviors. Ph.D. Dissertation (advisors: Mary Jean Harrold and James M. Rehg), December 2006.

[6] James A. Jones, James F. Bowring, and Mary Jean Harrold. Debugging in Parallel. Proceedings of the 2007 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2007), pages 16-26, July 2007.

[7] George K. Baah, Andy Podgurski, and Mary Jean Harrold. The Probabilistic Program Dependence Graph and Its Application to Fault Diagnosis Proceedings of the 2008 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), pages 189–200, July 2008.